

GO-05 07: Асинхронность. sync.Pool

Описание:

Самые затруднительные операции для нашего кода - это работа с памятью. Причем любой. Аллокация памяти занимает много места и времени. Для оптимизации работы с памятью в go есть специальный инструмент - sync.Pool.

sync.Pool:

В любом современном языке программирования есть сборщик мусора. Он автоматически освобождает память от не используемых компонентов и функций. Сборщик мусора влияет на производительность. Но при этом выполняет полезную функцию освобождения памяти. Сборщик мусора (далее GC) не постоянно собирает мусор, а через определённые промежутки времени. В случае если ваш код выделяет память под некоторые структуры данных, а потом освобождает их — и так по кругу — это вызывает определённое давление на GC, в том числе заставляет runtime обратиться к ОС для выделения новой памяти. Давайте посмотрим на основные операции с sync.Pool.

```
import(  
    "sync"  
)  
  
var bytesPool = sync.Pool{  
    New: func() interface{} { return []byte{} },  
}
```

В приме выше сначала создаем новую структуру Pool.

Чтобы положить данные в Pool нужно выполнить следующую команду.

```
bytesPool.Put(ary)
```

И, соответственно, получить данные из Pool можно следующим образом:

```
nextAry := bytesPool.Get().([]byte)
```

Как примерно работает Pool: Он выделяет память и GC не сбрасывает ее. Если у вас происходит обработка большого количества одинаковых данных, вы можете обрабатывать их и класть в Pool. С помощью горутин забирать оттуда данные, и доставлять их в нужное место.

Вот простая обертка над Pool, которая позволит работать с ним более эффективно:

```
// получить
```

```
func getBytes() (b []byte) {  
    ifc := bytesPool.Get()  
    if ifc != nil {
```

```

        b = ifc.([]byte)
    }
    return
}
// положить
func putBytes(b []byte) {
    if cap(b) <= maxCap {
        b = b[:0] // сброс
        bytesPool.Put(b)
    }
}

```

Хороший пример использования пула: пакет `fmt`. Со 109-ой по 150-ую строку. Помните, `Pool` может и ухудшить производительность вашего кода, важно использовать `Pool` грамотно. `Pool` потокобезопасен, поэтому он и находится в пакете `sync`.

Задание:

Вам дана структура:

```

type Counter struct{
    A int
    B int
}

```

1. Ваша задача написать функцию, которая увеличивает оба поля на 1.
2. Написать два бенчмарка: один - с использованием `sync.Pool`, другой - без.
3. Посмотреть сколько аллокаций на операцию было в первом и втором случае.
4. прислать ссылку на репозиторий и скриншот с результатами бенчмарков.

Шаблон бенчмарка (с ними вы еще познакомитесь подробнее в модуле 06.05) выглядит следующим образом:

```

func BenchmarkWithoutPool(b *testing.B) {
    for i := 0; i < b.N; i++ {
        for j := 0; j < 10000; j++ {
            b.StopTimer(); /*ваша функция инкремента*/; b.StartTimer()
        }
    }
}

```

Запуск выполняется следующей командой:

```
go test -bench=. bench_test.go.
```